REMARKS

Claims 1-28 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 112, Second Paragraph, Rejection:

The Examiner rejected claims 1-7, 12 and 25-28 under 35 U.S.C. § 112, second paragraph, as indefinite. Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, the Examiner asserts that the phrase "computer executable code built in to said device" is unclear and "fails to qualify how code is 'built in' to a given device, and what constitutes executable code being "built in" to an arbitrary device'. Applicants respectfully traverse this rejection and assert that claim 1, when read in its entirety and in light of Applicants' specification is clear and easily understood by one skilled in the art. The Examiner seems to be ignoring that claim 1 further recites that the pre-generated message interface is constructed prior to runtime and that it is implemented by computer-executable code built in to said device during a code-build process for the device. A code-build process is a well-understood and fundamental concept in the art of computer science and software/code application development. Thus, one skilled in the art would easily understand the subject matter recited in claim 1.

In the Response to Arguments section of the Final Action, the Examiner submits that Applicants did not sufficiently address the issues presented in the prior Office Action. The Examiner states, "The Examiner notes that citations presented above seem to indicate computer instructions [software features] that are included with the (Java) operating system installed on the device. The limitation indicating 'during code-build process' does not sufficiently clarify the issue regarding the 'built-in' limitation because the code-build process is inherent for any application and may be performed on a static or dynamic schedule." This assertion by the Examiner is clearly incorrect. For example, for

applications that support dynamic code generation, or that support run-time interfaces for obtaining dynamic proxies, etc., while the underlying code generator or interface may be "built in", any code generated or obtained at run-time is clearly not "built in" prior to runtime. These concepts are readily understood by anyone of ordinary skill in the art. For example, Applicants' specification refers to embodiments in which message interfaces are generated during the build of embedded software, which is another well-understood and fundamental concept in software development (see, e.g., p. 14, lines 1-3). In fact, the Examiner's own remarks indicate his understanding of software (e.g., operating system software) that is installed on the device (e.g., "built in") and note that it would be clear that such "built in" software would not be built using a code-build process on a dynamic schedule (e.g., at runtime), as the Examiner suggests, but that it would be built prior to installation (e.g., on a static schedule, in the Examiner's words).

Applicants again assert that the combination of limitations recited in claim 1, *wherein said pre-generated message interface is implemented by computer-executable code built in to said device during a code-build process for the device, wherein the pre-generated message interface is constructed prior to runtime*, when read as a whole with the rest of the claims and when taken in light of Applicants' specification, would be easily understood by one skilled in the art.

For at least the reasons above, removal of the § 112, second paragraph, rejection of claim 1 is respectfully requested. Similar remarks also apply to claims 14 and 27.

Regarding claims 12, 25, 26 and 28, Applicants again note that none of these claims include the phrase, "computer executable code built in to said device" and thus the rejection of these claims is improper. The Examiner has not provided any basis for the § 112, second paragraph, rejection of claims 12, 25, 26 and 28.

According to M.P.E.P. 2173.02, "In reviewing a claim for compliance with 35 U.S.C. 112, second paragraph, the examiner must consider the claim as a whole." Moreover, M.P.E.P. 2173.02 also requires that the claim be considered "in light of the

specification." *See, also, Orthokinetics, Inc. v. Safety Travel Chairs, Inc.*, 806 F.2d 1565, 1576, 1 USPQ2d 1081, 1088 (Fed. Cir. 1986). Appellants also refer to the Memorandum of January 17, 2003 from the Deputy Commissioner for Patent Examination Policy regarding advance notice of changes to MPEP § 2173.02 clarifying Office policy with respect to rejections made under 35 U.S.C. § 112, second paragraph in view of the Supreme Court holding in *Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co.*, 122 S.Ct. 1831, 62 USPQ2d 1705 (2002). The Memorandum states that "some latitude in the manner of expression and the aptness of terms should be permitted even though the claim language is not as precise as the examiner might desire." The Memorandum also states that "if other modes of expression selected by applicants satisfy the statutory requirements of 35 U.S.C. § 112, second paragraph, but the examiner merely wants the applicant to improve the clarity or precision of the language used, the claim must not be rejected under 35 U.S.C. § 112, second paragraph." *See, also, Exxon Research & Eng'g Co. v. U.S.*, 60 USPQ2d 1272, 1276 (Fed. Cir. 2001) ("We have not insisted that claims be plain on their face in order to avoid condemnation for indefiniteness; rather, what we have asked is that the claims be amenable to construction, however difficult that task may be." "If the meaning of the claim is discernible, even though the task may be formidable and the conclusion may be one over which reasonable persons will disagree, we have held the claim sufficiently clear to avoid invalidity on indefiniteness grounds.").

Section 102(e) Rejection:

The Examiner rejected claims 1-4, 8-17 and 21-28 under 35 U.S.C. § 102(e) as being anticipated by Weschler (U.S. Patent 6,842,903). Applicants respectfully traverse this rejection for at least the reasons below.

Regarding claim 1, Weschler fails to disclose **receiving an address for a service within the distributed computing environment and linking the address to a pre-generated message interface for accessing the service**, contrary to the Examiner's contention. Weschler teaches a method for providing dynamic references between services in a computer system that allows one service to obtain a reference to another

service without requiring specific knowledge of the other service. In Weschler's system, pluggable modules may be used to provide additional functionality for a service and a service connector interface encapsulates the logic necessary to locate an instance of a particular service.

The Examiner cited column 8, lines 5 – 10. However, none of the cited passages, even considered in view of the rest of Weschler, discloses receiving an address for a service and linking that address to a pre-generated message interface for accessing the service. At column 8, lines 5 – 10, Weschler describes pluggable interfaces and pluggable modules that provide supporting functionality and implement program behavior for core profiling engine 201. Weschler describes that modules are "plugged in by specifying an initialization parameter" that "comprises an address or fully qualified path pointing to a location at which the plug-in module is stored." However, the address described by Weschler is the address from which the plug-in model is loaded when instantiated (see, Weschler, column 8, lines 11-20).

Additionally, Weschler fails to disclose linking the address to a pre-generated message interface for accessing the service. The Examiner cited column 4, lines 15 – 30 and column 6, lines 20- 30 of Weschler. The cited passage describe Weschler's use of pluggable interfaces supporting connections to plug-in service modules. Weschler teaches that client applications access core profile engine 203 via a profile services API 203. API 203 "enables client applications that have a corresponding interface to send messages to request profile services from core profile engine 201." Weschler also teaches that core profile engine 201 supports methods for "gaining a reference to plug-in services".

However, Weschler fails to describe linking an address for a service to a pre-generated message interface for accessing the service. Firstly, the Examiner equates the address at which a plug-in module is stored with the address for a service of applicant's claim. However, the address at which a plug-in module is stored is not linked to a pre-generated message interface for accessing the service. Instead, Weschler teaches that the

plug-in module may be stored locally or may not be stored on the same machine as core profile engine 201. In Weschler's system the plug-in module is loaded from the address at which it is stored and instantiated to provide additional functionality to core profile engine 201. Nowhere does Weschler mention linking the address at which a plug-in module is stored, which the Examiner equates to an address for a service, to a pre-generated message interface for accessing the service.

In the Response to Arguments section of the Final Action, the Examiner disagrees with Applicants' argument above and submits that Weschler disclosed (in column 8, lines 50-55) a mechanism through which the application can obtain a reference (the URL address) to the service and use it. The Examiner submits that the plug-in is used to generate an interface to the service, and the application casts to the interface (column 9, lines 20-25). The Examiner contends that since the plug-in module is associated with the address, Weschler disclosed *linking an address to a message interface*. Applicants respectfully disagree. First, the Examiner's citation in column 9 does not describe generating an interface to the service, as the Examiner suggests, but instead describes obtaining information about what interface the particular service version implements by issuing a getServiceInterface(version) method. In addition, Applicants submits that "casting to an interface" to use a service is clearly not the same as linking an address to a pre-generated message interface for accessing the service, as recited in claim 1. Casting refers to converting a programming entity (e.g., variable, parameter, object, etc.) from one type to another. Thus, Weschler teaches converting the service interface (or a portion of it, or parameters to the interface) from one programming type (e.g., int, char, double, etc.) to another. **Casting from one programming type to another clearly has nothing to do with linking an address to a pre-defined message interface for accessing a service.**

Applicants respectfully remind the Examiner that anticipation requires the presence in a single prior art reference disclosure of each and every limitation of the claimed invention, arranged as in the claim. M.P.E.P 2131; *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 221 USPQ 481, 485 (Fed. Cir. 1984). The

identical invention must be shown in as complete detail as is contained in the claims. *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). As discussed above, Weschler fails to disclose receiving an address for a service within the distributed computing environment and linking the address to a pre-generated message interface for accessing the service. Therefore, Weschler cannot be said to anticipate claim 1.

For at least the reasons above, the rejection of claim 1 is not supported by the cited art and removal thereof is respectfully requested. Similar remarks also apply to claims 14 and 27.

Regarding claim 2, Weschler fails to disclose a message interface of the message endpoint **verifying that the messages sent to the service comply with a message schema for the service.** The Examiner cited column 6, lines 25-50 of Weschler. However, the cited passage states that Weschler's core profile engine 201 provides a limited set of functions includes, among others, "management utilities for defining schemas." Weschler does not describe verifying that messages sent to the service comply with a message schema for the service. Weschler does not state that a schema managed via the functions provided by the core profile engine 201 is usable to verify messages sent to the service. Instead, Weschler teaches that core profile engine 201 provides management utilities for *defining* schemas. The fact that Weschler's core profile engine 201 provides functions for defining schemas has nothing do with, and fails to disclose, verifying that messages sent to a service comply with a message schema for the service.

Applicants again assert that a single mention of "management utilities for defining schemas", as cited by the Examiner, does not in any way disclose the specific limitation of verifying that message sent to a service comply with a message schema for the service.

In the Response to Arguments section of the Final Action, the Examiner cites Weschler in disclosing that the response message is sent back through API 203 to the

appropriate protocol adapter 205 (or built-in adapter 205) to the requesting client application 202 (column 7, lines 18-20.) The Examiner contends that Weschler disclosed the verification limitation of claim 2 "because determining the appropriate protocol adapter would have inherently included verification for compliance with the message schema for the service." Applicants respectfully disagree. First, there is nothing in Weschler that describes determining the appropriate protocol, or how such a determination may be made. The only reference to setting a protocol when exchanging messages is found in column 9, lines 13-14, "Because the service 306 provider also provides and programs the service connector 304, the protocol for effectuating this is set at that time." This passage clearly does not disclose that the protocol is determined from a message itself, as the Examiner seems to imply, or that a message is (or should be) analyzed to see if it complies with a particular protocol (or schema) for this or any other reason. In fact, since the response message is sent from core engine 201 and may be generated by core engine 201 to be correct (i.e., compliant to the same protocol used for the request message) by construction. Applicants assert that none of the features of the system disclosed in Weschler necessarily require verification of compliance to a message schema.

"To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is **necessarily present** in the things described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.'" *In re Robertson*, 169 F.3d 743, 745, 49 USPA2d 1949, 1950-51 (Fed. Cir. 1999) (emphasis added). Applicants assert that it would clearly not be necessary in the system of Weschler to verify compliance with a particular message schema for a given service in order to determine an appropriate protocol adapter, to send a response message to an appropriate protocol adapter, or to perform any other functions described in Weschler.

For at least the reasons above, the rejection of claim 2 is not supported by the cited art and removal thereof is respectfully requested. Similar remarks also apply to claims 9, 15, and 22.

Regarding claim 8, Weschler fails to disclose **receiving a schema defining messages for accessing the service and generating message endpoint code according to the schema**. In the previous Response, Applicants noted that Weschler teaches a method for providing dynamic references between services in a computer system that allows one service to obtain a reference to another service without requiring specific knowledge of the other service and that Weschler teaches a service connector interface that encapsulates the logic necessary to locate an instance of a particular service.

In the Response to Arguments section of the Final Action, the Examiner cites column 6, lines 30-35 "gaining a reference to data store adapters" as teaching *receiving a schema defining messages for accessing the service*. The Examiner also cites column 9, lines 20-25 "the application casts to the interface" and column 8, lines 60-65 "service connector may be compiled along with the application" as teaching *generating message endpoint code according to said schema* and *linking said message endpoint code into executable operating code for the device*. The Examiner submits that data store adapters would inherently involve a schema for accessing the data structures involved. First, Applicants note that the Examiner's citation is column 6 describes gaining references to plug-in services, some of which may be data store adapters. Even if these data store adapters involved a schema, there is nothing in Weschler that describes receiving a schema defining messages for accessing the data store, as recited in claim 8. Furthermore, the Examiner's references to casting to an interface and compiling a service connector also do not teach generating code (i.e., *generating message endpoint code*) according to the schema, as recited in claim 8.

Applicants also note that the Examiner does not cite anything in Weschler that teaches *loading the message endpoint code* (i.e., message endpoint code generated according to the message schema) *and operating code onto the device*, nor does Weschler teach this limitation of claim 8.

Thus, for at least the reasons presented above, the rejection of claim 8 is not supported by the cited art and removal thereof is respectfully requested. Similar remarks also apply to claim 14 and 27.

Section 103(a) Rejection:

The Examiner rejected claims 1-4, 8-17 and 21-28 under 35 U.S.C. § 103(a) as being unpatentable over Roberts et al. (U.S. Patent 6,560,633) (hereinafter "Roberts") in view of Chen et al. (U.S. Publication 2002/0062334) (hereinafter "Chen"). Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, Roberts in view of Chen fails to teach or suggest **linking said address to a pre-generated message interface for accessing said service, wherein said pre-generated message interface is implemented by computer-executable code built in to said device during a code-build process for the device, wherein the pre-generated message interface is constructed prior to runtime**. Roberts teaches that the interfaces relied upon by the Examiner are downloaded from a web services engine and used by a client application during runtime (see, Roberts, column 4, line 60 – column 5, line 7, column 5, lines 19-30, and column 7, lines 10-33). Specifically, Roberts teaches a runtime model that "draws on the use of a number of web services that construct special user interface pages as output data" and that "[t]he behavior of these pages is to generate subsequent web service requests to the web services engine, and to call for the execution [of] action defined in a web services application session" (Roberts, column 4, line 67 – column 5, line 5). For example, Roberts describes that a web services request, "generates a response that takes the form of a graphical user interface to be displayed in a browser". Roberts WSA interfaces are clearly meant to be downloaded and constructed *at runtime* and thus teach away from the use of computer-executable code built in to a device and comprising a pre-generated message interface for accessing a service, where the pre-generated message interface is constructed *prior* to runtime.

The Examiner relies on Chen to teach the use of dynamic agents to provide dynamic behavior modification of agents. Chen's agents are designed to carry application specific actions, which can be loaded and modified on the fly (e.g. at runtime). Chen's dynamic behavior modification allows a dynamic agent to adjust its capability for accommodating environmental and requirement changes. Chen's dynamic agents may also play different roles across multiple applications. The Examiner cites paragraph [0063] and refers to Chen's built-in APIs. However, the cited passage of Chen, even if combined with Roberts, does not teach or suggest computer-executable code built in to the device comprises the pre-generated message interface. Instead, Chen teaches built-in APIs that allows an action to create "a receiver thread and [to] register[] its socket address" (Chen, paragraph [0063]). APIs for creating threads and registering a socket address do not, even if combined with Roberts, teach or suggest a computer-executable code built into a device during a code-build process for the device and that comprises a pre-generated message interface for accessing the service.

In fact, nowhere does Roberts or Chen, whether considered singly or in combination teach or suggest linking said address to a pre-generated message interface for accessing said service, wherein said pre-generated message interface is implemented by computer-executable code built in to said device during a code-build process for the device, wherein the pre-generated message interface is constructed prior to runtime. Thus, even if combined as suggested by the Examiner, Roberts and Chen fail to teach or suggest the limitations of claim 1. Instead, the combination of Roberts and Chen, as suggested by the Examiner would result in a system using the WSA interfaces of Roberts but that also includes built in APIs for creating receiver threads and registering socket addresses.

In the Response to Arguments section of the Final Action, the Examiner disagrees with Applicants' statement, "Roberts WSA interfaces are clearly meant to be downloaded and constructed *at runtime."* The Examiner submits that Roberts disclosed a pre-generated message interface was constructed prior to runtime in column 13, lines 15-20 "templates build the program prior to running." Applicants assert that the Examiner is

ignoring the specific language recited in claim 1. Claim 1 does not just recite that a pre-generated message interface is constructed prior to running, but prior to runtime (i.e., before any code is run.) In addition, claim 1 recites that the pre-generated message interface is implemented by computer-executable code built in to said device during a code-build process for the device. The WSA interfaces of Roberts are clearly not built in to the devices on which they run prior to runtime. Instead, they are "child runtime models" dynamically constructed at runtime according to lists of features contained in templates (see, e.g., Roberts, column 10, lines 14-25). Therefore, the use of these templates clearly does not teach or suggest the limitations of claim 1.

For at least the reasons above, the rejection of claim 1 is not supported by the cited art and removal thereof is respectfully requested. Similar remarks also apply to claims 14 and 27.

Regarding claim 8, the Examiner has again failed to provide a proper rejection under § 103 of this claim. The Examiner again lists claim 8 as rejected under 35 U.S.C. 103(a), but fails to provide an actual rejection. Claim 8 recites subject matter different from that recited by claim 1, the only claim for which the Examiner provides a rejection under 35 U.S.C. 103(a). Therefore, the rejection of claim 8 under § 103 is improper.

Further regarding claim 8, and as noted in Applicants' previous Response, Roberts in view of Chen fails to teach or suggest *pre-generating at least one message interface to be built-in to a device in order to access a service* by *receiving a schema defining messages for accessing the service; generating message endpoint code according to said schema; and linking said message endpoint code into executable operating code for the device and loading the message endpoint code and operating code onto the device*.

In the Response to Arguments section of the Final Action, the Examiner disagrees with Applicants' previous assertion that Roberts and Chen do not teach linking message endpoint code, generated *according to a schema* defining messages for accessing a service, *into executable operating code* for a device. The Examiner cites Roberts (column 7, lines 10-15 and lines 45-50) as teaching a regeneration process for a transformed runtime model and fully interactive user interfaces, and that the runtime models follow a schema. Applicants assert that this general reference to "a schema" does not teach the limitations of claim 8, which recite that message endpoint code (i.e., code for a pre-defined message interface) is generated according to a schema defining messages for accessing a service. Applicants assert that "a schema" for a runtime model may specify many different things, and that it may or may not have anything to do with defining messages for accessing a service. Nothing in Roberts discloses that this particular type of schema is used in generating the child runtime models.

In addition, the regeneration process of building a runtime child model clearly does not teach pre-generating at least one message interface to be built-in to a device to access a service, as recited in claim 8. Instead, the regeneration process of Roberts is performed at runtime. Similarly, the user interfaces are described as "a set of dynamically generated and updated XML entities called Pages" in column 7, lines 11-13 (emphasis added.) Therefore, these child runtime models and user interfaces are clearly not pre-generated message interfaces that are generated according to the limitations of claim 8 and linked into executable operating code for a device.

Additionally, Chen, whether considered singly or in combination with Roberts, also fails to teach or suggest generating message endpoint code according to a schema defining messages for accessing a service, linking the message endpoint code into executable operating code for the device and loading the message endpoint code and operating code onto the device. The Examiner has not relied upon Chen or cited any portion of Chen regarding this limitation of claim 8. Thus, the Examiner's combination of Roberts and Chen clearly fails to teach or suggest the limitations of claim 8.

Furthermore, as noted in Applicants' previous Response, Roberts specifically teaches downloading WSA interfaces from web service devices for use on client devices to access WSAs. Downloading of message interface is clearly quite different from linking message endpoint code, generated according to a schema defining messages for accessing a service, *into executable operating code* for a device and loading the message endpoint code and operating code onto the device. Thus, **Roberts teaches away** from Applicants' claim.

For at least the reasons above, the rejection of claim 8 is not supported by the cited art and removal thereof is respectfully requested. Similar remarks also apply to claims 21 and 28.

Claims Objected To But Otherwise Allowable:

In the Office Action mailed August 21, 2006, the Examiner object to claims 5-7 and 18-20 as being dependent upon a rejected base claim, but stated they would be allowable if rewritten in independent form. As these claims were not specifically addressed in the Final Action, Applicants assume that their status has not changed. In light of the above remarks, Applicants again assert that these claims are allowable in their present form.

CONCLUSION

Applicants submit the application is in condition for allowance, and prompt notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5181-66200/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicants

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: March 26, 2007